Lematizador Morfosintáctico y Semántico Robusto con Flexionador y Estimador Idiomático, usando algoritmos eficientes y compactos para idiomas muy ricos en formas como el español

Andrés Tomás Hohendahl [A]
andres.hohendahl@fi.uba.ar

José Francisco Zelasco [A,B]
jfzelasco@fi.uba.ar

[A] Laboratorio de Estereología y Mecánica Inteligente.
Dto. de Ing. Mecánica Facultad de Ingeniería,
Universidad de Buenos Aires, Paseo Colón 850,
(1065), Buenos Aires, Argentina.
[B] INTIA, Facultad de Ciencias Exactas, UNCPBA,
Universidad del Centro de la Provincia de Buenos Aires,
Campus Universitario, Paraje Arroyo Seco
(B7000) Tandil, Prov. de Buenos Aires, Argentina.

Abstract

We present a word recognition and generation system for multilingual natural language processing, intended for human-machine interface. Presenting robust, low memory footprint and efficient algorithms, it is capable of: language identification, linguistic word-tagging, semantic extraction, automatic error recognition and correction with morphologic and sound-like estimation capability. It uses simple rules to express sophisticated and reversible morphological changes. Tolerates, detects and corrects spelling errors, primarily intended for text generated by automatic natural speech and writing recognition; constrained inputs like mobile phone keyboards or PDA's, chat and/or e-mails. Useful for interactive text correction & assistance in word processing, it yields a low memory footprint and high processing speed, being adequate for personal computers, portables, palms, mobiles & embedded solutions. For spanish, it needs 200Kb for 50k lemmas and 4500 rules, equivalent to 1.2M exact words and >300M guessable. Capable of morphological and sound-like inference, in a similar way as a natural language human hearer would perform. As flexion generator, it has added semantic expression capability.

Keywords: Natural Language Processing, NLP, Semantic, Morphologic, Efficient Algorithm, Low Memory Footprint, Part of Speech Tagging, POS.

Resumen

Se presenta un sistema de reconocimiento y flexión de palabras en lenguaje natural orientado a interfase hombre-máquina. Presentamos algoritmos robustos, eficientes y con poca impronta de memoria, capaces de realizar identificación idiomática, etiquetado lingüístico, extracción semántica, estimación morfológica y acústica (por similitud). Usa reglas simples capaces de expresar sofisticados cambios morfológicos reversibles. Tolera, detecta y corrige errores, estando principalmente orientado a textos provenientes de reconocimiento automático de voz y texto escrito, mensajes de teclados restringidos como terminales móviles "sms/mms/wap", PDA's etc.., "chat" y/o

e-mail. Es apropiado para asistencia y corrección interactiva en procesamiento de texto, tiene baja impronta de memoria y alta velocidad de proceso, siendo adecuado para ordenadores personales, portátiles, móviles y productos embebidos. Para el español, requiere 200Kb para 50k lemas y 4500 reglas, equivalentes a 1.2M palabras exactas y >300M estimables. Puede inferir por similitud morfológica y tónica, en forma similar a la de un hablante natural. Como flexionador posee además capacidad de expresión semántica.

Palabras claves: Procesamiento de Lenguaje Natural, PNL, Flexionador, Morfología, Semántica, Poca Memoria, Algoritmo Eficiente, Etiquetador de partes de la Oración, Idioma.

INTRODUCCIÓN

1. El Diccionario Lingüístico

En momentos en donde la computación está tan difundida y hay sobrada capacidad de cálculo, aún quedan pendientes cosas que hace décadas se suponía que estarían resueltas en estos días. Una de las más sorprendentes es la comunicación hombre-máquina en lenguaje natural. Esta asignatura pendiente aún no ha salido de los laboratorios académicos y una de las razones de esto es la enorme complejidad que conlleva. La ingeniería lingüística es la rama que investiga y desarrolla estas herramientas y actualmente es una industria en pleno desarrollo. La investigación en esta dirección trae muchas alternativas y opciones posibles en cada una, pero una de las piedras angulares es la clasificación para el ingreso de datos inicial, básicamente la búsqueda en diccionario.

Aquí es precisamente en donde fallan mayormente los sistemas, puesto que la complejidad de algunas lenguas es inmensa, casi inabordable. Es una costumbre de hoy día en temas de computación y en especial a nivel académico, utilizar recursos sin una mesura real y final del alcance y aplicabilidad del desarrollo. Este es precisamente el motivo por el cual sistemas de reconocimiento lingüístico descansan sobre enormes bases de datos lingüísticos, los cuales están solamente disponibles en centros de estudios especializados y cuyas licencias son complejas y costosas. Además estos datos cambian constantemente conforme aparecen nuevas acepciones y se perfeccionan otras, lo cual implica una constante actualización.

Uno de los motivos de las dificultades en esta área es, precisamente por la enorme cantidad de datos que deben estar almacenados y la ambigüedad de los mismos y sus estructuras, presentes cada nivel sucesivo de análisis. En el inglés, las flexiones de los verbos son apenas 4 por verbo, los sustantivos y adjetivos solamente existen en dos formas (plural y singular), los artículos son pocos y no poseen género ni número. En cambio en el español, solamente la flexión de los verbos involucra: 3 personas, 2 números 4 tiempos, 3 modos y ciertas combinaciones especiales, resultando cerca de 70 formas verbales simples, si a esto anexamos las formas compuestas y los pronombres enclíticos anidados, tenemos varias centenas de posibles palabras y grupos a considerar por cada verbo. Con los sustantivos y adjetivos, suceda algo similar si consideramos los sufijos y prefijos diminutivos, aumentativos, peyorativos y otros, además del plural,

singular y algunas veces, el neutro, el colectivo (para grupos). Como es de esperar, la dimensión de estas cifras es casi inmanejable sin algoritmos de reducción o bases de datos enormes. Un diccionario completo para el español que contenga las formas mas usadas, tendría más de 300 millones de palabras [25], ocupando alrededor de 2-5 Terabytes, dependiendo del método de almacenamiento y la cantidad de información contenida por palabra. Esto es calculado asumiendo palabras de 3.5 letras como promedio y etiquetas (contenido) simple: lema, clasificación gramatical, persona, modo, tiempo, género y número. Esto no involucra información semántica ni de relaciones entre palabras como ser sinónimos, antónimos y lemas.

2. Almacenamiento y Recuperación de Datos

La respuesta a este tipo de problemas de gran cantidad de datos, puede parecer simple, existiendo técnicas de compresión que casi llegan al almacenamiento óptimo. El verdadero problema no es como compactar espacio, sino que para poder usarlos para búsquedas, hay que tenerlos disponibles rápidamente y en estructuras adecuadas (ej.: árboles). Las bases de datos que usan SQL, son la solución generalmente escojida por ambientes académicos y empresas comerciales. Estas, suelen poseer tamaños importantes de instalación, costos y estructuras de licencias complejas y un alto consumo de recursos, no resultando apropiadas para búsquedas parciales, palabras con letras cambiadas, faltantes o insertadas. Sus estructuras de índices internas tampoco son las apropiadas para búsquedas parciales o por comienzo y terminación de palabras, puesto que las búsquedas SQL usan algoritmos optimizados y heurísticas que son poco eficientes para estos casos. El resultado con estas bases, es que se usan demasiados recursos, para búsquedas relativamente simples. Queda por analizar la posibilidad de organizar todos los datos en memoria que es un recurso limitado y costoso. Para los sistemas de 32 bits hogareños y actuales hay un límite práctico de 2.0 Gb de RAM y esto no alcanzaría para contener una base de datos razonable para cubrir el español, en un formato abierto.

La solución que hallamos, es el poder representar los datos en memoria en una forma ingeniosamente mínima, tanto respecto a su estructura de índices para asegurar búsquedas de análisis como también para poder usarlas en búsquedas inversas, puesto que el sistema tiene ambas finalidades. Otras implementaciones de paquetes con clasificación morfo-sintáctica y semántica para Natural Lenguaje Processing (NLP), como ser el FreeLing[30] si bien no documentan el uso de memoria y recursos, hemos determinado el costo computacional de proceso y es varios ordenes de magnitud mayor al nuestro pues debe leer alrededor de 50 Mb antes de cada proceso. Tampoco posee la posibilidad de tolerar errores y presentar palabras alternativas “similares”,tampoco ofrece detectar/inferir el idioma. Existen recientemente documentadas, algunas otras aplicaciones para el español [33] [34] de las cuales no hemos podido acceder a las cifras de perfomance ni al tamaño de sus bases de datos.

3. Reglas para prefijos y sufijos

Las reglas de flexión para prefijos y sufijos, se han diseñado basándonos en un estándar de reglas llamado compresión AFFIX [13] ampliado para permitir recuperación y expresión gramática y semántica.

Este formato expresa la síntesis de una flexión, incluyendo las modificaciones que sufre la raíz o forma canónica al flexionarse. La notación propuesta puede expresar patrones de coincidencia muy sofisticadas usando expresiones regulares [14] simples, expresando la transformación en modo directo, fácilmente entendible. Se ha diseñado un algoritmo eficiente y reversible para ambos sentidos: el análisis y la síntesis.

Flexionando, transforma la raíz canónica en la flexión, aplicando la regla y en modo de análisis, conociendo la regla y la palabra flexionada se pueda hallar la raíz canónica. Se diseñó un mecanismo alternativo, en donde dada una raíz y un conjunto de condiciones gramaticales, junto a condiciones semánticas, el sistema es capaz de armar una palabra flexionada resultante.

Esto proporciona una capacidad de generar palabras "sintéticas" que pueden no pertenecer al diccionario, en especial cuando se usan prefijos y sufijos en formas no verbales (médicos, latinos, griegos, superlativos, diminutivos, peyorativos). Esta capacidad de síntesis es similar a la de un hablante natural, quien conoce los afijos por su función semántica y los aplica para modificar o colorear el significado de una palabra o concepto, ignorando si la misma está o no algún diccionario.

3.1. Formato de Almacenamiento de Reglas

El algoritmo usa las palabras almacenadas en su forma canónica, seguida de un indicador del conjunto de reglas son aplicables para flexionarla, usando un símbolo luego de un separador '/'. El número de reglas parciales se han agrupado por función gramática, tanto para prefijación como para sufijación, resultando en un número lo suficientemente pequeño para poder usar una sola letra (menos de 64 símbolos).
Este mecanismo, al expresar el tipo de cambio o accidente gramatical que sufre la palabra raíz permite que durante el proceso de análisis, se obtenga una clasificación inmediata. La misma puede involucrar aspectos tanto gramaticales como semánticos.
El mecanismo de expresión posee la capacidad de que una palabra posea diferentes funciones o etiquetas para su raíz que en sus grupos de flexiones, para esto se prevee una serie de etiquetas, adicionadas luego de las reglas, deparados por un separador '/'. Luedo del mismo posee opcionalmente una etiqueta de caracterización fonológica (suena como), con el mismo tipo de separador '/'.

Para permitir una construcción simple de las reglas, se utilizó el formato de ISPELL [14] el cual permite una expresión muy clara y didáctica ala hora de sintetizar las reglas, permitiendo además comentarios. Se creó un módulo capaz de importar este formato, verificando su integridad y generando luego el formato más compacto de representación llamado "affix".

Las palabras raíz, en el formato "affix" se representan del siguiente modo:

```
      AMAR/XYV/VMN/3M2R

línea ::= palabra [/[reglas]/[etiquetas][/sound] [#comentario]<CR>
reglas ::= Letra1 [ Letra2..]
etiquetas ::= Etiqueta1 [, Etiqueta2..]
sound ::= Sonido1 [,Sonido2..]
```

Lo cual indica que la raíz es **AMAR** y las reglas aplicables son "**X**" "**Y**" y "**V**"; su etiqueta "**VMN**" indica: verbo regular infinitivo y suena como "**3M2R**". Todas son optativas.
Veamos una la regla "**V**" en detalle, expresada en formato ISPELL:
(NOTA: todo lo que sigue al "**#**" es considerado comentario, salvo las meta-funciones a principio de línea: **#GT, #GR, #GS y #GP**)

```
[suffix]
flag *V: # Verbos de todas las conjugaciones regulares en
PRESENTE
#GS verbo regular infinitivo
#GR verbo regular
#GT PRESENTE INDICATIVO
#GP primera singular
    A R           > -AR, O          # amar amo
    [^CG] E R     > -ER, O          # comer como
    C E R         > -CER, ZO        # vencer venzo
    G E R         > -GER, JO        # coger cojo
    [^CGU] I R    > -IR, O          # vivir vivo
    C I R         > -CIR, ZO        # esparcir esparzo
    G I R         > -GIR, JO        # fingir finjo
    G U I R       > -UIR, O         # distinguir distingo
    Q U I R       > -QUIR, CO       # delinquir delinco
```

En este ejemplo se observa que la primera línea **[suffix]** caracteriza la sección para describir reglas de sufijación, luego **flag *V:** indica el nombre de la regla "**V**". Le suceden una serie de líneas comenzando con letras o grupos, separados por espacios. Cada uno de ellos expresa una terminación, en el orden correspondiente, la cual debe coincidir con la terminación de la palabra raíz en cuestión. Para poder ser aplicada, en este caso a palabra **AMAR**, la regla que coincidirá será la primera. El símbolo "**>**" indica la transformación, luego se sucede el **> -AR**, que indica que hay que quitar esa partícula, luego "**, O**" indica lo que hay que agregar, una sola letra en este caso para transformar **AMAR** en **AMO**. Veamos el segundo ejemplo, aquí se ve una expresión regular **[^CG]** la cual indica que en esa posición es admitida toda letra que no sea ni "**C**" o "**G**". Hay que tener especial cuidado de no expresar una regla de modo que letra a "quitar" coincida con una expresión regular, puesto que inmediatamente se vuelve no reversible (en forma única) y simple al algoritmo, puesto que en este caso habría que generar el conjunto complementario a la expresión regular para luego hallar el conjunto de palabras que pudo sufrir la transformación, tarea un tanto compleja. Cabe destacar que entre las palabras en forma canónica o raíz y su forma flexionada pueden no haber ninguna raíz en común, tal es el caso del verbo irregular **IR** cuyo pasado indicativo en primera persona es **FUI** y su gerundio es **YENDO.**

4. Funciones Adicionales

Las meta-funciones **#GP #GT #GR #GS** han sido agregadas en forma de comentario sin perder compatibilidad con cualquier sistema anterior que pueda usar las reglas creadas y al mismo tiempo poder expresar cómodamente atributos de las transformaciones "flexiones" en forma acumulativa. Estas luego serán traducidas a una forma no acumulativa en un formato AFFIX ampliado para contener este tipo de información. Todas estas meta-funciones son válidas desde su aparición dentro de cada regla.

```
#GS   Elementos semánticos agregado a la forma canónica
#GR   Características de base para toda la regla
#GT   Adicionales a la base (#GR), elimina acumulaciones (#GP y #GT)
#GP   Formato Parcial no acumulable, adicionado a (#GR y #GT)
```

Esta nomenclatura, se compila en tiempo de importación del archivo *.aff, acumulando las reglas como lista de palabras, adicionadas a cada sub-regla. Estas palabras forman un conjunto el cual se usa en un formato de compresión por diccionario, resultando en unas pocas letras en formato compactado, al final de cada regla en formato AFFIX.

Este formato incluye posibilidad muti-alternativa, es decir si una transformación (regla) es aplicable tanto a sustantivos comunes como a adjetivos calificativos, la regla se traduce como:

```
#GR sustantivo común | adjetivo calificativo
```

Esto permite una flexibilidad adicional para expresar reglas en forma compacta, puesto que prácticamente toda palabra a la que esta regla es aplicable, será considerada tanto sustantivo como adjetivo, generando la lógica multiplicidad de etiquetas.

5. Lematizador: Mecanismo de Análisis

El sistema para reconocer las flexiones que ha sufrido un lema, llamado lematizador, se basa en aplicar una a una las reglas de flexión en forma inversa (des-flexión) hallando una presumible raíz o forma canónica. Luego a ésta forma se la busca entre las palabras del diccionario con sus reglas. Si es hallada se coteja que posea la regla desde la cual se partió y en caso positivo, ésta raíz es la forma canónica buscada.

Durante este proceso se obtiene acumulativamente, un conjunto de etiquetas, tanto la regla aplicada, como la palabra hallada pueden tener etiquetas gramaticales, mientras que las reglas además pueden además contener datos semánticos. Estas reglas se combinan mediante un mecanismo acumulativo con reglas propias de prioridad que aseguran que se construirá la o las etiquetas gramaticales apropiadas y de haberla, también la alguna información semántica. El resultado se expresa en una estructura de datos en formato EAGLES 2.0 con el adicionado de un conjunto de palabras especiales para uso semántico.

5. Implementación de Reglas

El mecanismo original, propuesto por ASPELL[13] y NetSpell [16], se prueban todas y cada una de las reglas de flexión contra el diccionario completo de las palabras, buscando en un diccionario basados en dispersión (Hashtable) hasta hallar una o ninguna coincidencia. Esto denota si tenemos NR número de reglas y NP de palabras una cantidad equivalente de operaciones (des-flexión y búsqueda): NR*NP. Además en la mayoría de los diccionarios de uso libre para formato ASPELL hay una importante cantidad de palabras sin etiquetas (entre 20 y 30%), entre nombres propios y formas no verbales, los cuales abultan la búsqueda sin esperar resultado alguno. Este método funciona en un modo que está muy de moda desde que el recurso de computación se ha vuelto tan popular y veloz, que llamaríamos “fuerza bruta”.

6. Mejoras Obtenidas

El mecanismo propuesto, como veremos a continuación economiza mucho el recurso computacional al tiempo que por utilizar estructuras óptimas, permite reducir el número de operaciones significativamente, en un orden de más de 100 veces, respecto a sistemas tradicionales, trasformando mecanismos de búsqueda de datos con tiempos promedio **O**(n log(n)) a mecanismos lineales **O**(p). Es importante destacar que el mecanismo de búsqueda, no es proporcional al número de palabras/reglas "n" en el diccionario, sino que en nuestros algoritmos, solamente es sensible linealmente al número de letras "p" de la palabra a hallar. Esto lo hace inmensamente poderoso, puesto que el tiempo del proceso es independiente de la riqueza o tamaño de la base de datos morfológicos. Además el proceso internamente "mientras" halla la/las posibles flexiones para hallar la raíz o lema, construye la lista de las "parecidas" en forma transparente y sin agregar costo de proceso extra.

Primero se agrega una sintaxis adicional al sistema ASPELL, y es el de poder anidar en una misma regla terminaciones optativas, esto reduce el universo de expresión de reglas, dividiéndolo por el un número de reglas que se combinen en cada grupo, por ejemplo a las reglas de flexión verbales, las redujo en promedio de 2 a 4 veces y en algunos casos de pronombres enclíticos anidados hasta 30 veces. Esto no agrega tiempo de proceso en el análisis de los cambios morfológicos y flexión, y de paso agrega flexibilidad y fácil expresión a la hora de expresar la regla.

Veamos como se implementó para una regla llamada "**T**"

```
flag *T:    # Verbos transitivos con gerundio regular + enclítico
#GS verbo transitivo infinitivo
#GR verbo transitivo
#E1  LO     Singular Masculino      # dormir > dormírmelo
#E1  LA     Singular Femenino       # dormir > dormírmela
#E1  LE     Singular Neutro         # dormir > dormírmele
#E1  LOS    Plural Masculino        # dormir > dormírmelos
#E1  LAS    Plural Femenino         # dormir > dormírmelas
#E1  LES    Plural Neutro           # dormir > dormírmeles
#GT infinitivo enclítico
     [AEI] R      ?
#GT gerundio enclítico
     A R    ? -AR, ÁNDO
     E R    ? -ER, IÉNDO
     I R    ? -R, ÉNDO
```

Este ejemplo ilustra la regla anidada "**#E1**" la cual es aplicable a todas aquellas terminaciones en donde el separador es "**?**" en lugar de "**>**" este separador indica que esta sub-regla, utiliza todas las posibles combinaciones indicadas "**#E1**" al principio de la regla. El mecanismo prevé la posibilidad de expresar mas profundidad de anidación puesto que el "1" que sigue la letra "E" indica que ese es el grupo de anidación. En la práctica no resultó necesario puesto que eran muy pocas las reglas que lo requerían y el tiempo de proceso extra sería contraproducente, pero el formato se plantea abierto y

luego del signo de pregunta se pueden especificar una secuencia de números que denotan las sub-reglas y su orden de aplicación.

El resultado de esta regla compilada es el siguiente en formato ASPELL:

```
T Y 10 O5-mmA
1*LO Opcw
1*LA Opc4
1*LE Opeg
1*LOS Opsw
1*LAS Ops4
1*LES Opug
T1 0 0 [aei]r PJ-mnqA
T1 ar ándo ar PJ-mlqA
T1 er iéndo er PJ-mlqA
T1 r éndo ir PJ-mlqA
```

Se observa claramente la correspondencia entre ambos formatos, obviamente el ASPELL es mas compacto pero resulta complejo para escribirlo o editarlo directamente. Las secuencias "**O5-mmA**" son el formato compactado con diccionario de las palabras: "**verbo transitivo infinitivo**" al igual que "**Opcw**" significa "**Singular Masculino**" y para las reglas la sigla "**PJ-mnqA**" simboliza "**infinitivo enclítico**".

6.1 Estructuras Optimizadas

El mecanismo completo que hemos utilizado en la implementación, difiere radicalmente del propuesto, primeramente se utiliza una estructura de árbol "Trie" [17] reversible, para almacenar los afijos terminaciones/comienzos transformadas con capacidad de múltiples elementos por nodo, esto permite que las reglas sean seleccionadas directamente en tantas comparaciones como letras tenga el afijo, esto reduce de 4500 a apenas 3.2 comparaciones promedio, para esa regla (1400:1). Para no complicar el algoritmo, se cicla por las 30 reglas globales con un lazo for-next el cual es extremadamente rápido y no justificaba ingresar los datos en un Trie unificado, simplificando el armado de las estructuras de datos en memoria durante la carga del diccionario. Dentro de cada regla "flag" hay 2 Tries, uno directo y otro reverso, ambos orientados a agilizar la selección de reglas a aplicar/invertir, basado en la palabra a analizar/flexionar. Para el caso de estructuras anidadas, se utiliza un Trie unificado el cual guarda óptimamente todas las combinaciones, asegurando la velocidad de acceso lineal O(n), conforme al máximo de "n" letras de la terminación en cada regla.

Para el almacenamiento y búsqueda de palabras en formato canónico o raíz, decidimos utilizar una variante de Ternary Serach Tree [18], siendo éste un mecanismo con performance similar y a veces hasta mejor que una tabla de diccionario por dispersión (*hashtable*), y una impronta de memoria similar y ligeramente mayor. Los beneficios de este mecanismo son que permite hallar palabras con cambios de 1..N letras y con caracteres de búsqueda, es decir encontrar palabras similares a otras con determinado patrón o cota de medida de distancia entre palabras.

Se implementó una variante de búsqueda que permite buscar independientemente de los

acentos y eñes, para permitir hallar raíces aproximadas prescindiendo de los diacríticos y generar alternativas en caso de no hallar la palabra deseada al lematizarla. Esto arroja resultados muy buenos ya que la longitud de las búsquedas es a lo sumo de N+P comparaciones, donde N es el número de letras de la palabra a buscar y P es el número de "errores" o variantes permitidas.

6. Formatos de Diccionarios

Para el etiquetado de palabras se hizo una modificación adicional permitiendo etiquetado acumulativo, para evitar numerosas etiquetas repetidas y permitir agrupar palabra bajo un mismo rótulo, lo cual resulta muy útil a la hora de agregar palabras y editar la lista de las mismas. El formato ampliado es muy sencillo de entender y se mantiene total compatibilidad con el formato anterior; sólo se agregan 2 prefijos correspondiendo * a las etiquetas gramatical y % al de reglas.

Los mecanismos de compresión ideados resultan muy eficientes, tanto a la hora de levantar el archivo a memoria como en el tamaño total del mismo, generándose diccionarios inferiores en tamaño a los utilizados por Open-Office (OO) [16], aún ampliando su capacidad expresiva y de reconocimiento en varias veces. El archivo en texto plano para el español: "es-ES.dic" de OO pesa 712k con 48k palabras mientras que el nuestro es de 545k con 3.5k palabras más, agregando 526 reglas gramaticales con clasificación gramática. (~200 k compactado)

7. Medición de Velocidad

Medido en una PC, i386 Celeron-D 2.66GHz, 512k RAM, XP y .NET 1.1 el sistema es capaz de etiquetar, lematizando 2700 palabras por segundo, entregando en cada caso una clase instanciada con su lista de palabras similares, con sus etiquetas EAGLES 2.0 y extensión semántica. En cambio procesa más de 53.000 palabras por segundo en búsqueda aproximada con errores gramaticales (diacríticos). Esto realizado sobre un corpus de 2666 palabras de 5.02 letras en promedio texto: "Verdad y Perspectiva de Ortega y Gasset". Ocupando apenas 12 Mb de memoria RAM, incluyendo el desperdicio asociado a la tecnología de 32 bits respecto a la alineación de 4 bytes. La tasa de detección de errores sintácticos es del 89.64% y la tasa de reconocimiento gramatical exitoso fue de 38% para este texto ya que contiene una importante cantidad de palabras y nombres propios. El sistema fue utilizado con un diccionario con ~53.000 formas canónicas y ~4500 reglas de flexión. La velocidad de estimación de idioma, para Alemán, Inglés y Español, fue de 21.300 palabras por segundo. Procesando el Diccionario etiquetado de ~70.600 palabras del diccionario de FreeLing1.4, el sistema halla ~70.000 etiquetas exactamente, las 600 no halladas, analizadas en detalle revelan errores del diccionario (conjugación errónea) y no de etiquetado.

8. Conclusiones

El sistema presentado, posee numerosas virtudes, entre las cuales una de las mas importantes y que ha sido la intención del diseño, es la capacidad de manejar diferentes niveles de reconocimiento, desde exactos hasta aproximados, brindando una serie de palabras alternativas "similares" en todos los casos, para una posible evaluación adicional por parte del sistema siguiente en la cadena de proceso NLP.

Otra ventaja es la incorporación de un detector idiomático [22], pudiendo discernir de pleno palabras no pertenecientes a tal o cual idioma, con un costo de apenas un par de sumas proporcional al número de letras "n" de la palabra por el número de idiomas "l" propuestos/analizados, siendo su complejidad apenas **O**(p*l).
La última ventaja es su velocidad y bajo consumo de recursos computacionales, tanto en proceso como de almacenamiento, lo hacen ideal para aplicaciones embebidas del mundo real: módulos "plug-ins" para procesadores de texto, reconocedores de lenguaje, traductores, sistemas de comandos y diálogo, búsqueda de respuestas y otros que requieran procesamiento lingüístico complejo y veloz.

8.1. Aplicaciones

Los sistemas portátiles, los de uso cotidiano y hogareño como las PC's, PDA's, Tablet-PC y Teléfonos Celulares, se están convirtiendo en elementos de uso común y frecuente, con una oferta tecnológica creciente, tanto en diversidad como en prestaciones.
Una de las principales desventajas de esta explosión de tecnología y prestaciones es precisamente la complejidad de las mismas y en especial la interfase con el usuario. El procesamiento de texto natural, siempre ha sido la ciencia que abordó estos problemas, con la contrapartida de los enormes desafíos planteados por la dificultad de representar mecanismos "naturales" como el habla y la comprensión mediante software.
Estas aplicaciones no son posibles sin sistemas de este tipo que aportan la posibilidad de poder procesar y reconocer palabras tanto en forma precisa como aproximada, incluyendo el idioma, junto al poder sintetizar las mismas mediante información gramatical y semántica simultáneas. Todo esto realizado en forma eficiente y con poco uso de memoria, permitirá sin duda la incorporación de herramientas con algoritmos lingüísticos a dispositivos portátiles y de uso cotidiano con limitaciones de memoria y procesamiento.
Estos dispositivos, muñidos de sistemas de procesamiento de lenguaje natural, mejorarán sin duda nuestra calidad de vida y nos ayudarán cada vez mas y mejor con las tareas cotidianas. Mas precisamente la posibilidad de poseer mecanismos de síntesis y reconocimiento de formas canónicas eficientes y veloces permite toda una nueva clase de aplicaciones en donde estos mecanismos pueden formar parte de lazos de realimentación y búsqueda para hallar soluciones óptimas rápidamente unidos con algoritmos mucho mas complejos como los probabilísticos, heurísticos entre otros tantos utilizados en Inteligencia Artificial, la resolución de problemas y relaciones, búsqueda de anáforas, extracción de información, hallazgo de entidades (NER: Named Entity Recognition), entre otras muchas aplicaciones en donde la interfaz lingüística es tanto el primero como el último de los eslabones.
Finalmente, la capacidad de poder flexionar palabras tanto exacta como aproximadamente lo hace candidato a ser utilizado en síntesis de sentencias de lenguaje natural para diálogo y confección de respuestas, basado en gramáticas y semánticas generativas.

9. Palabras Finales

La computación, como ya se manifiesta claramente, en un muy futuro cercano se volverá inevitablemente más y más abarcativa, poderosa, veloz compacta y eficiente. Este devenir de virtudes, cuya gran mayoría son derivadas de la aplicación intensiva de mejoras tecnológicas en los procesos de fabricación, debe estar acompañada de prestaciones cada vez mas amigables al usuario. Esto en definitiva se expresará en la

posibilidad de dialogar directamente con los sistemas en lenguaje natural, con el menor esfuerzo por nuestra parte. Va con esto nuestro aporte de una arquitectura de algoritmos eficientes en pos de esta meta.

Biblilografía

[16] Open Office Dictionaries: http://lingucomponent.openoffice.org/spell_dic.html

[13] Affix compression: http://aspell.sourceforge.net/man-html/Affix-Compression.html

[14] Expresiones Regulares: http://www.regular-expressions.info/

[17] Estructuras de árboles "Trie": http://www.nist.gov/dads/HTML/trie.html

[18] Estructuras de árboles Ternary Serach Tree:
http://www.nist.gov/dads/HTML/ternarySearchTree.html
[22] Algoritmos eficientes para detección temprana de errores y clasificación idiomática para uso en procesamiento de lenguaje natural y texto. Andres T. Hohendahl & José F. Zelasco, WICC2006 - http://www.unimoron-wicc2006.com.ar/ ISBN 950-9474-35-5
[25] Diccionarios españoles: http://www3.unileon.es/dp/dfh/jmr/dicci/012.htm

[30] FreeLing, un proyecto de software libre para NLP: www.lsi.upc.es/~nlp/**freeling**/

[33] FLANOM: Flexionador y lematizador automático de formas nominales.
Santana, O.; Pérez, J.; Carreras, F.; Duque, J.; Hernández, Z.; Rodríguez, G.
Lingüística Española Actual XXI, 2, 1999. Ed. Arco/Libros, S.L.
253/297

[34] FLAVER: Flexionador y lematizador automático de formas verbales.
Santana, O.; Pérez, J.; Hernández, Z.; Carreras, F.; Rodríguez, G.
Lingüística Española Actual XIX, 2, 1997. Ed. Arco/Libros, S.L.
229/282

[36] http://www.gedlc.ulpgc.es/investigacion/desambigua/morfosintactico.htm

[37] Relaciones morfoléxicas prefijales del español. Santana, O.; Carreras, F.; Pérez, J.; Rodríguez, G. Boletín de Língüística, Vol. 22. ISSN: 0798-9709. Jul/Dic, 2004. 79/123.